CHAPTER 3

# MICROSOFT BASIC FUNCTIONS

The intrinsic functions provided by Microsoft BASIC are presented in this chapter. The functions may be called from any program without further definition.

Arguments to functions are always enclosed in parentheses. In the formats given for the functions in this chapter, the arguments have been abbreviated as follows:

| | |
|---|---|
| X and Y | Represent any numeric expressions |
| I and J | Represent integer expressions |
| X$ and Y$ | Represent string expressions |

If a floating point value is supplied where an integer is required, BASIC will round the fractional portion and use the resulting integer.

NOTE

With the Microsoft BASIC interpreter, only integer and single precision results are returned by functions. Double precision functions are supported only by the Microsoft BASIC Compiler.

## 3.1 ABS

Format: ABS(X)

Action: Returns the absolute value of the expression X.

Example:

```
PRINT ABS(7*(-5))
 35
Ok
```

## 3.2 ASC

Format: ASC(X$)

Action: Returns a numerical value that is the ASCII code of the first character of the string X$. (See Appendix D for ASCII codes.) If X$ is null, an "Illegal function call" error is returned.

Example:

```
10 X$ = "TEST"
20 PRINT ASC(X$)
RUN
 84
Ok
```

See the CHR$ function for ASCII-to-string conversion.

## 3.3 ATN

Format: ATN(X)

Action: Returns the arctangent of X in radians. Result is in the range -pi/2 to pi/2. The expression X may be any numeric type, but the evaluation of ATN is always performed in single precision.

Example:

```
10 INPUT X
20 PRINT ATN(X)
RUN
? 3
 1.24905
Ok
```

## 3.4 CDBL

Format: CDBL(X)

Action: Converts X to a double precision number.

Example:

```
10 A = 454.67
20 PRINT A;CDBL(A)
RUN
 454.67  454.6700134277344
Ok
```

## 3.5 CHR$

Format: CHR$(I)

Action: Returns a string whose one element has ASCII code I. (ASCII codes are listed in Appendix D.) CHR$ is commonly used to send a special character to the terminal. For instance, the BEL character could be sent (CHR$(7)) as a preface to an error message, or a form feed could be sent (CHR$(12)) to clear a terminal screen and return the cursor to the home position.

Example:

```
PRINT CHR$(66)
B
Ok
```

See the ASC function for ASCII-to-numeric conversion.

## 3.6 CINT

Format: CINT(X)

Action: Converts X to an integer by rounding the fractional portion. If X is not in the range -32768 to 32767, an "Overflow" error occurs.

Example:

```
PRINT CINT(45.67)
 46
Ok
```

See the CDBL and CSNG functions for converting numbers to the double precision and single precision data type. See also the FIX and INT functions, both of which return integers.

## 3.7 COS

Format: COS(X)

Action: Returns the cosine of X in radians. The calculation of COS(X) is performed in single precision.

Example:

```
10 X = 2*COS(.4)
20 PRINT X
RUN
 1.84212
Ok
```

## 3.8 CSNG

Format: CSNG(X)

Action: Converts X to a single precision number.

Example:

```
10 A# = 975.3421#
20 PRINT A#; CSNG(A#)
RUN
 975.3421  975.342
Ok
```

See the CINT and CDBL functions for converting numbers to the integer and double precision data types.

## 3.9 CVI, CVS, CVD

Format:

```
CVI(<2-byte string>)
CVS(<4-byte string>)
CVD(<8-byte string>)
```

Action: Convert string values to numeric values. Numeric values that are read in from a random disk file must be converted from strings back into numbers. CVI converts a 2-byte string to an integer. CVS converts a 4-byte string to a single precision number. CVD converts an 8-byte string to a double precision number.

Example:

```
 .
 .
 .
70 FIELD #1,4 AS N$, 12 AS B$, ...
80 GET #1
90 Y=CVS(N$)
 .
 .
 .
```

See also MKI$, MKS$, MKD$, Section 3.25 and PART II, Chapter 3, Microsoft BASIC Disk I/O, of the Microsoft BASIC User's Guide.

## 3.10 EOF

Format:

```
EOF(<file number>)
```

Action: Returns -1 (true) if the end of a sequential file has been reached. Use EOF to test for end-of-file while INPUTting, to avoid "Input past end" errors.

Example:

```
10 OPEN "I",1,"DATA"
20 C=0
30 IF EOF(1) THEN 100
40 INPUT #1,M(C)
50 C=C+1:GOTO 30
 .
 .
 .
```

## 3.11 EXP

Format: EXP(X)

Action: Returns e to the power of X. X must be <=87.3365. If EXP overflows, the "Overflow" error message is displayed, machine infinity with the appropriate sign is supplied as the result, and execution continues.

Example:

```
10 X = 5
20 PRINT EXP (X-1)
RUN
 54.5982
Ok
```

## 3.12 FIX

Format: FIX(X)

Action: Returns the truncated integer part of X. FIX(X) is equivalent to SGN(X)*INT(ABS(X)). The major difference between FIX and INT is that FIX does not return the next lower number for negative X.

Examples:

```
PRINT FIX(58.75)
 58
Ok

PRINT FIX(-58.75)
-58
Ok
```

## 3.13 FRE

Format:

```
FRE(0)
FRE(X$)
```

Action: Arguments to FRE are dummy arguments. FRE returns the number of bytes in memory not being used by BASIC.

FRE("") forces a garbage collection before returning the number of free bytes. BE PATIENT: garbage collection may take 1 to 1-1/2 minutes. BASIC will not initiate garbage collection until all free memory has been used up. Therefore, using FRE("") periodically will result in shorter delays for each garbage collection.

Example:

```
PRINT FRE(0)
 14542
Ok
```

## 3.14 HEX$

Format:

```
HEX$(X)
```

Action: Returns a string which represents the hexadecimal value of the decimal argument. X is rounded to an integer before HEX$(X) is evaluated.

Example:

```
10 INPUT X
20 A$ = HEX$(X)
30 PRINT X "DECIMAL IS " A$ " HEXADECIMAL"
RUN
? 32
 32 DECIMAL IS 20 HEXADECIMAL
Ok
```

See the OCT$ function for octal conversion.

## 3.15 INKEY$

Format: INKEY$

Action: Returns either a one-character string containing a character read from the terminal or a null string if no character is pending at the terminal. No characters will be echoed and all characters are passed through tto the program except for Control-C, which terminates the program. (With the Microsoft BASIC Compiler, Control-C is also passed through to the program.)

Example:

```
1000 'TIMED INPUT SUBROUTINE
1010 RESPONSE$=""
1020 FOR I%=1 TO TIMELIMIT%
1030 A$=INKEY$ : IF LEN(A$)=0 THEN 1060
1040 IF ASC(A$)=13 THEN TIMEOUT%=0 : RETURN
1050 RESPONSE$=RESPONSE$+A$
1060 NEXT I%
1070 TIMEOUT%=1 : RETURN
```

## 3.16 INP

Format: INP(I)

Action: Returns the byte read from port I. I must be in the range 0 to 255. INP is the complementary function to the OUT statement, Section 2.47.

Example:

```
100 A=INP(255)
```

## 3.17 INPUT$

Format: INPUT$(X[,[#]Y])

Action: Returns a string of X characters, read from the terminal or from file number Y. If the terminal is used for input, no characters will be echoed and all control characters are passed through except Control-C, which is used to interrupt the execution of the INPUT$ function.

Example 1:

```
5 'LIST THE CONTENTS OF A SEQUENTIAL FILE IN
HEXADECIMAL
10 OPEN"I",1,"DATA"
20 IF EOF(1) THEN 50
30 PRINT HEX$(ASC(INPUT$(1,#1)));
40 GOTO 20
50 PRINT
60 END
```

Example 2:

```
   .
   .
   .
100 PRINT "TYPE P TO PROCEED OR S TO STOP"
110 X$=INPUT$(1)
120 IF X$="P" THEN 500
130 IF X$="S" THEN 700 ELSE 100
   .
   .
   .
```

## 3.18 INSTR

Format: INSTR([I,]X$,Y$)

Action: Searches for the first occurrence of string Y$ in X$ and returns the position at which the match is found. Optional offset I sets the position for starting the search. I must be in the range 1 to 255. If I>LEN(X$) or if X$ is null or if Y$ cannot be found, INSTR returns 0. If Y$ is null, INSTR returns I or 1. X$ and Y$ may be string variables, string expressions or string literals.

Example:

```
10 X$ = "ABCDEB"
20 Y$ = "B"
30 PRINT INSTR(X$,Y$);INSTR(4,X$,Y$)
RUN
 2  6
Ok
```

NOTE: If I=0 is specified, error message "ILLEGAL ARGUMENT IN <line number>" will be returned.

## 3.19 INT

Format: INT(X)

Action: Returns the largest integer <=X.

Examples:

```
PRINT INT(99.89)
 99
Ok

PRINT INT(-12.11)
-13
Ok
```

See the FIX and CINT functions which also return integer values.

## 3.20 LEFT$

Format: LEFT$(X$,I)

Action: Returns a string comprised of the leftmost I characters of X$. I must be in the range 0 to 255. If I is greater than LEN(X$), the entire string (X$) will be returned. If I=0, the null string (length zero) is returned.

Example:

```
10 A$ = "BASIC"
20 B$ = LEFT$(A$,5)
30 PRINT B$
BASIC
Ok
```

Also see the MID$ and RIGHT$ functions.

## 3.21 LEN

Format: LEN(X$)

Action: Returns the number of characters in X$. Non-printing characters and blanks are counted.

Example:
```
10 X$ = "PORTLAND, OREGON"
20 PRINT LEN(X$)
 16
Ok
```

## 3.22 LOC

Format: LOC(<file number>)

Action: With random disk files, LOC returns the record number just read or written from a GET or PUT. If the file was opened but no disk I/O has been performed yet, LOC returns a 0. With sequential files, LOC returns the number of sectors (128 byte blocks) read from or written to the file since it was OPENed.

Example:
```
200 IF LOC(1)>50 THEN STOP
```

## 3.23 LOG

Format: LOG(X)

Action: Returns the natural logarithm of X. X must be greater than zero.

Example:

```
PRINT LOG(45/7)
 1.86075
Ok
```

## 3.24 LPOS

Format: LPOS(X)

Action: Returns the current position of the line printer print head within the line printer buffer. Does not necessarily give the physical position of the print head. X is a dummy argument.

Example:

```
100 IF LPOS(X)>60 THEN LPRINT CHR$(13)
```

## 3.25 MID$

Format: MID$(X$,I[,J])

Action: Returns a string of length J characters from X$ beginning with the Ith character. I and J must be in the range 1 to 255. If J is omitted or if there are fewer than J characters to the right of the Ith character, all rightmost characters beginning with the Ith character are returned. If I>LEN(X$), MID$ returns a null string.

Example:

```
LIST
10 A$="GOOD "
20 B$="MORNING EVENING AFTERNOON"
30 PRINT A$;MID$(B$,9,7)
Ok
RUN
GOOD EVENING
Ok
```

Also see the LEFT$ and RIGHT$ functions.

NOTE: If I=0 is specified, error message "ILLEGAL ARGUMENT IN <line number>" will be returned.

## 3.26 MKI$, MKS$, MKD$

Format:

```
MKI$(<integer expression>)
MKS$(<single precision expression>)
MKD$(<double precision expression>)
```

Action: Convert numeric values to string values. Any numeric value that is placed in a random file buffer with an LSET or RSET statement must be converted to a string. MKI$ converts an integer to a 2-byte string. MKS$ converts a single precision number to a 4-byte string. MKD$ converts a double precision number to an 8-byte string.

Example:

```
90 AMT=(K+T)
100 FIELD #1, 8 AS D$, 20 AS N$
110 LSET D$ = MKS$(AMT)
120 LSET N$ = A$
130 PUT #1
  .
  .
```

See also CVI, CVS, CVD, Section 3.9 and Microsoft BASIC Disk I/O, in the Microsoft BASIC User´s Guide.

## 3.27 OCT$

Format: OCT$(X)

Action: Returns a string which represents the octal value of the decimal argument. X is rounded to an integer before OCT$(X) is evaluated.

Example:

```
PRINT OCT$(24)
 30
Ok
```

See the HEX$ function for hexadecimal conversion.

## 3.28 PEEK

Format: PEEK(I)

Action: Returns the byte (decimal integer in the range 0 to 255) read from memory location I. I must be in the range 0 to 65536. PEEK is the complementary function to the POKE statement, Section 2.48.

Example:

```
A=PEEK(&H5A00)
```

## 3.29 POS

Format: POS(I)

Action: Returns the current cursor position. The leftmost position is 1. X is a dummy argument.

Example:
```
IF POS(X)>60 THEN PRINT CHR$(13)
```

Also see the LPOS function.

## 3.30 RIGHT$

Format: RIGHT$(X$,I)

Action: Returns the rightmost I characters of string X$. If I=LEN(X$), returns X$. If I=0, the null string (length zero) is returned.

Example:
```
10 A$="DISK BASIC"
20 PRINT RIGHT$(A$,8)
RUN
BASIC
Ok
```

Also see the MID$ and LEFT$ functions.

## 3.31 RND

Format: RND[(X)]

Action: Returns a random number between 0 and 1. The same sequence of random numbers is generated each time the program is RUN unless the random number generator is reseeded (see RANDOMIZE, Section 2.53). However, X<0 always restarts the same sequence for any given X.

X>0 or X omitted generates the next random number in the sequence. X=0 repeats the last number generated.

Example:

```
10 FOR I=1 TO 5
20 PRINT INT(RND*100);
30 NEXT
RUN
 24  30  31  51  5
Ok
```

## 3.32 SGN

Format: SGN(X)

Action: If X>0, SGN(X) returns 1.
If X=0, SGN(X) returns 0.
If X<0, SGN(X) returns -1.

Example: ON SGN(X)+2 GOTO 100,200,300 branches to 100 if X is negative, 200 if X is 0 and 300 if X is positive.

## 3.33 SIN

Format: SIN(X)

Action: Returns the sine of X in radians. SIN(X) is calculated in single precision. COS(X)=SIN(X+3.14159/2).

Example:

```
PRINT SIN(1.5)
 .997495
Ok
```

## 3.34 SPACE$

Format: SPACE$(X)

Action: Returns a string of spaces of length X. The expression X is rounded to an integer and must be in the range 0 to 255.

Example:

```
10 FOR I = 1 TO 5
20 X$ = SPACE$(I)
30 PRINT X$;I
40 NEXT I
RUN
  1
   2
    3
     4
      5
Ok
```

Also see the SPC function.

## 3.35 SPC

Format: SPC(I)

Action: Prints I blanks on the terminal. SPC may only be used with PRINT and LPRINT statements. I must be in the range 0 to 255. A´;´ is assumed to follow the SPC(I) command.

Example:

```
PRINT "OVER" SPC(15) "THERE"
OVER               THERE
Ok
```

Also see the SPACE$ function.

## 3.36 SQR

Format: SQR(X)

Action: Returns the square root of X. X must be >=0.

Example:

```
10 FOR X = 10 TO 25 STEP 5
20 PRINT X, SQR(X)
30 NEXT
RUN
 10            3.16228
 15            3.87298
 20            4.47214
 25            5
Ok
```

## 3.37 STR$

Format: STR$(X)

Action: Returns a string representation of the value of X.

Example:

```
5 REM ARITHMETIC FOR KIDS
10 INPUT "TYPE A NUMBER";N
20 ON LEN(STR$(N)) GOSUB 30,100,200,300,400,500
     .
     .
     .
```

Also see the VAL function.

## 3.38 STRING$

Formats:

```
STRING$(I,J)
STRING$(I,X$)
```

Action: Returns a string of length I whose characters all have ASCII code J or the first character of X$.

Example:

```
10 X$ = STRING$(10,45)
20 PRINT X$ "MONTHLY REPORT" X$
RUN
----------MONTHLY REPORT----------
Ok
```

## 3.39 TAB

Format: TAB(I)

Action: Spaces to position I on the terminal. If the current print position is already beyond space I, TAB goes to that position on the next line. Space 1 is the leftmost position, and the rightmost position is the width minus one. I must be in the range 1 to 255. TAB may only be used in PRINT and LPRINT statements.

Example:

```
10 PRINT "NAME" TAB(25) "AMOUNT" : PRINT
20 READ A$,B$
30 PRINT A$ TAB(25) B$
40 DATA "G. T. JONES","$25.00"
RUN
NAME                    AMOUNT

G. T. JONES             $25.00
Ok
```

## 3.40 TAN

Format: TAN(X)

Action: Returns the tangent of X in radians. TAN(X) is calculated in single precision. If TAN overflows, the "Overflow" error message is displayed, machine infinity with the appropriate sign is supplied as the result, and execution continues.

Example:

```
10 Y = Q*TAN(X)/2
```

## 3.41 USR

Format : USR[<digit>](X)

Action: Calls the user´s assembly language subroutine with the argument X. <digit> is in the range 0 to 9 and corresponds to the digit supplied with the DEF USR statement for that routine. If <digit> is omitted, USR0 is assumed. See Assembly Language Subroutines, in the Microsoft BASIC User´s Guide.

Example:

```
40 B = T*SIN(Y)
50 C = USR(B/2)
60 D = USR(B/3)
    .
    .
    .
```

## 3.42 VAL

Format: VAL(X$)

Action: Returns the numerical value of string X$. The VAL function also strips leading blanks, tabs, and linefeeds from the argument string. For example,

```
VAL(" -3)
```

returns -3.

Example:

```
10 READ NAME$,CITY$,STATE$,ZIP$
20 IF VAL(ZIP$)<90000 OR VAL(ZIP$)>96699 THEN
PRINT NAME$ TAB(25) "OUT OF STATE"
30 IF VAL(ZIP$)>=90801 AND VAL(ZIP$)<=90815 THEN
PRINT NAME$ TAB(25) "LONG BEACH"
    .
    .
    .
```

See the STR$ function for numeric to string conversion.

## 3.43 VARPTR

Format 1: VARPTR(<variable name>)

Format 2: VARPTR(#<file number>)

Action: Format 1: Returns the address of the first byte of data identified with <variable name>. A value must be assigned to <variable name> prior to execution of VARPTR. Otherwise an "Illegal function call" error results. Any type variable name may be used (numeric, string, array), and the address returned will be an integer in the range 32767 to -32768. If a negative address is returned, add it to 65536 to obtain the actual address.

VARPTR is usually used to obtain the address of a variable or array so it may be passed to an assembly language subroutine. A function call of the form VARPTR(A(0)) is usually specified when passing an array, so that the lowest-addressed element of the array is returned.

NOTE: All simple variables should be assigned before calling VARPTR for an array, because the addresses of the arrays change whenever a new simple variable is assigned.

Format 2: For sequential files, returns the starting address of the disk I/O buffer assigned to <file number>. For random files, returns the address of the FIELD buffer assigned to <file number>.

Example:

```
100 X=USR(VARPTR(Y))
```

# APPENDIX A

## Summary of Error Codes and Error Messages

| Code | Number | Message |
|---|---|---|
| NF | 1 | NEXT without FOR<br>A variable in a NEXT statement does not correspond to any previously executed, unmatched FOR statement variable. |
| SN | 2 | Syntax error<br>A line is encountered that contains some incorrect sequence of characters (such as unmatched parenthesis, misspelled command or statement, incorrect punctuation, etc.). |
| RG | 3 | Return without GOSUB<br>A RETURN statement is encountered for which there is no previous, unmatched GOSUB statement. |
| OD | 4 | Out of data<br>A READ statement is executed when there are no DATA statements with unread data remaining in the program. |
| FC | 5 | Illegal function call<br>A parameter that is out of range is passed to a math or string function. An FC error may also occur as the result of:<br>1. a negative or unreasonably large subscript<br>2. a negative or zero argument with LOG<br>3. a negative argument to SQR<br>4. a negative mantissa with a non-integer exponent |

5. a call to a USR function for which the starting address has not yet been given

6. an improper argument to MID$, LEFT$, RIGHT$, INP, OUT, WAIT, PEEK, POKE, TAB, SPC, STRING$, SPACE$, INSTR, or ON...GOTO.

| | | |
|---|---|---|
| OV | 6 | Overflow<br>The result of a calculation is too large to be represented in Microsoft BASIC´s number format. If underflow occurs, the result is zero and execution continues without an error. |
| OM | 7 | Out of memory<br>A program is too large, has too many FOR loops or GOSUBs, too many variables, or expressions that are too complicated. |
| UL | 8 | Undefined line<br>A line reference in a GOTO, GOSUB, IF...THEN...ELSE or DELETE is to a nonexistent line. |
| BS | 9 | Subscript out of range<br>An array element is referenced either with a subscript that is outside the dimensions of the array, or with the wrong number of subscripts. |
| DD | 10 | Redimensioned array<br>Two DIM statements are given for the same array, or a DIM statement is given for an array after the default dimension of 10 has been established for that array. |
| /0 | 11 | Division by zero<br>A division by zero is encountered in an expression, or the operation of involution results in zero being raised to a negative power. Machine infinity with the sign of the numerator is supplied as the result of the division, or positive machine infinity is supplied as the result of the involution, and execution continues. |
| ID | 12 | Illegal direct<br>A statement that is illegal in direct mode is entered as a direct mode command. |
| TM | 13 | Type mismatch<br>A string variable name is assigned a numeric value or vice versa; a function that expects a numeric argument is given a string argument |

| | | |
|---|---|---|
| | | or vice versa. |
| OS | 14 | Out of string space<br>String variables have caused BASIC to exceed the amount of free memory remaining. BASIC will allocate string space dynamically, until it runs out of memory. |
| LS | 15 | String too long<br>An attempt is made to create a string more than 255 characters long. |
| ST | 16 | String formula too complex<br>A string expression is too long or too complex. The expression should be broken into smaller expressions. |
| CN | 17 | Can´t continue<br>An attempt is made to continue a program that:<br>1. has halted due to an error,<br>2. has been modified during a break in execution, or<br>3. does not exist. |
| UF | 18 | Undefined user function<br>A USR function is called before the function definition (DEF statement) is given. |
| | 19 | No RESUME<br>An error trapping routine is entered but contains no RESUME statement. |
| | 20 | RESUME without error<br>A RESUME statement is encountered before an error trapping routine is entered. |
| | 21 | Unprintable error<br>An error message is not available for the error condition which exists. This is usually caused by an ERROR with an undefined error code. |
| | 22 | Missing operand<br>An expression contains an operator with no operand following it. |
| | 23 | Line buffer overflow<br>An attempt is made to input a line that has too many characters. |
| | 26 | FOR without NEXT |

A FOR was encountered without a matching NEXT.

29 WHILE without WEND
A WHILE statement does not have a matching WEND.

30 WEND without WHILE
A WEND was encountered without a matching WHILE.

## Disk Errors

50 Field overflow
A FIELD statement is attempting to allocate more bytes than were specified for the record length of a random file.

51 Internal error
An internal malfunction has occurred in Microsoft BASIC. Report to Microsoft the conditions under which the message appeared.

52 Bad file number
A statement or command references a file with a file number that is not OPEN or is out of the range of file numbers specified at initialization.

53 File not found
A LOAD, KILL or OPEN statement references a file that does not exist on the current disk.

54 Bad file mode
An attempt is made to use PUT, GET, or LOF with a sequential file, to LOAD a random file or to execute an OPEN with a file mode other than I, O, or R.

55 File already open
A sequential output mode OPEN is issued for a file that is already open; or a KILL is given for a file that is open.

57 Disk I/O error
An I/O error occurred on a disk I/O operation. It is a fatal error, i.e., the operating system cannot recover from the error.

61 Disk Full

# Appendix B

# DISPLAY DRIVER SPECIFICATIONS

## INTRODUCTION B.1

The software in the BIOS display interface receives ASCII characters and either displays them or uses them for display control. Characters in the ASCII control set (hex 00 through hex 1F and hex 7F) are control characters and are not displayed (see the ESC 8 exception below). These characters may, however, affect the display. Most of the control characters act independently. But for some actions, more than one control character is required to specify the action. This is done by using the ASCII escape code (hex 1B) followed by one or more characters. The control characters and the escape sequences are described in B.2 and B.3.

The display controls are accessed from high-level languages by using the PRINT or equivalent statements. For example, in MS-BASIC—

**PRINT CHR$(27)+CHR$(112)**

turns on reverse video.

NOTE: Decimal 27 equals hex 1B, which is Escape, and decimal 112 is hex 70.

## CONTROL CHARACTERS B.2

▶ Bell (ALT-G, hex 07): This is not really a display control character. It sends the CODEC a series of signals that cause it to make the sound of a bell.

- ▶ Backspace (ALT-H, hex 08): Moves the cursor one column to the left. If cursor is at column 1, it moves to column 80 of the previous row (unless cursor is at column 1, row 1 in which case it moves to column 80, row 1).
- ▶ Horizontal Tab (ALT-I, hex 09): Moves the cursor to the next tab stop. Tab stops are fixed at columns 9, 17, 25, 33, 41, 49, 57, 65, and 72 through 80. If the cursor is at column 80, it remains there.
- ▶ Line Feed (ALT-J, hex 0A): Moves the cursor down one row. If cursor is at row 24, then the display scrolls up one row. (The Line feed can also be treated as a Return. See ESC x9.)
- ▶ Carriage Return (ALT-M, hex 0D): Moves the cursor to column 1 of the current row. (Return can also be treated as a Line feed. See ESC x8.)
- ▶ Shift In (ALT-N, hex 0E): Shifts to character set 1 (G1).
- ▶ Shift Out (ALT-0,hex 0F): Shifts to character set 0 (G0).

## B.3 ESCAPE SEQUENCES

**Table B-1: Cursor Functions**

| ESCAPE SEQUENCE | ASCII CODE GENERATED (HEXADECIMAL) | SEQUENCE DEFINITION |
|---|---|---|
| Esc-A | 1B, 41 | Moves the cursor up one line. |
| Esc-B | 1B, 42 | Moves the cursor down one line without changing columns. |
| Esc-C | 1B, 43 | Moves the cursor one character position to the right. |

continued

| ESCAPE SEQUENCE | ASCII CODE GENERATED (HEXADECIMAL) | SEQUENCE DEFINITION |
|---|---|---|
| Esc-D | 1B, 44 | Moves the cursor one character position to the left. |
| Esc-H | 1B, 48 | Moves the cursor to the home position (upper-left corner of screen). |
| Esc-I | 1B, 49 | Moves the cursor up one line, and stays in the same column. |
| Esc-n | 1B, 6E | Reports the cursor position. |
| Esc-j | 1B, 6A | The display driver saves the cursor position. |
| Esc-k | 1B, 6B | Returns the cursor to the previously-saved cursor position. |
| Esc-Y[l][c] | 1B, 59 | Moves the cursor via direct cursor addressing, where l is the line number (hex) and c is the column number (hex). The first line and the left column are both 20 hex (the smallest value of the printing characters) and increase from there. Since the lines are numbered from 1 to 19 hex (from top to bottom) and the columns from 1 to 50 hex (from left to right), you must add the proper line and column numbers to 1F. No movement occurs if l and/or c are invalid. |

## Table B-2: Editing Functions

| ESCAPE SEQUENCE | ASCII CODE GENERATED (HEXADECIMAL) | SEQUENCE DEFINITION |
|---|---|---|
| Esc-E | 1B, 45 | Erases the entire screen. |
| Esc-b | 1B, 62 | Erases from the start of the screen up to (and including) the cursor position. |
| Esc-J | 1B, 4A | Erases from the cursor position to the end of the page. |
| Esc-l | 1B, 6C | Erases entire line. |
| Esc-o | 1B, 6F | Erases from the beginning of line up to (and including) the cursor position. |
| Esc-K | 1B, 4B | Erases from cursor position to the end of the line. |
| Esc-L | 1B, 4C | Inserts a blank line. The current line and all following lines scroll down one line. The cursor moves to the beginning of the blank line. |
| Esc-M | 1B, 4D | Moves cursor to beginning of line, deletes the line, and then scrolls all following lines up one line. A blank line is inserted at line 24. |
| Esc-N | 1B, 4E | Deletes character at cursor position and shifts the rest of the line one character position to the left. |
| Esc-@ | 1B, 40 | Enters the Insert Character mode. This lets you insert characters into screen text. As each new character is inserted, the character at the end of the line is lost. |
| Esc-O | 1B, 4F | Exits Insert Character mode. |

## Table B-3: Configuration Functions

| ESCAPE SEQUENCE | ASCII CODE GENERATED (HEXADECIMAL) | SEQUENCE DEFINITION | |
|---|---|---|---|
| Esc-x[Ps] | 1B, 78 | Sets mode(s) as follows: | |
| | | Ps | Mode |
| | | 1 | Enable 25th line. |
| | | 3 | Hold screen mode on. |
| | | 4 | Use block cursor. |
| | | 5 | Cursor off. |
| | | 8 | Automatic line feed after a Return. |
| | | 9 | Automatic Return after a line feed. |
| | | A | Increase audio volume. |
| | | B | Increase CRT brightness. |
| | | C | Increase CRT contrast. |
| Esc-y[Ps] | 1B, 79 | Resets mode(s) as follows: | |
| | | Ps | Mode |
| | | 1 | Disable 25th line. |
| | | 3 | Hold screen mode off. |
| | | 4 | Use underscore cursor. |
| | | 5 | Cursor on. |
| | | 8 | No auto line feed. |
| | | 9 | No auto Return. |
| | | A | Decrease audio volume. |
| | | B | Decrease CRT brightness. |
| | | C | Decrease CRT contrast. |
| Esc-^ | 1B, 5E | Toggle hold mode. | |
| Esc-[ | 1B, 5B | Set hold mode. | |
| Esc-\ | 1B, 5C | Clear hold mode. | |
| Esc-\| | 1B, 7C | Activate user-defined console (T.B.A.). | |

## Table B-4: Operation Mode Functions

| ESCAPE SEQUENCE | ASCII CODE GENERATED (HEXADECIMAL) | SEQUENCE DEFINITION |
|---|---|---|
| Esc-p | 1B, 70 | Enters reverse video mode. |
| Esc-q | 1B, 71 | Exits reverse video mode. |

## Table B-5: Special Functions

| ESCAPE SEQUENCE | ASCII CODE GENERATED (HEXADECIMAL) | SEQUENCE DEFINITION |
|---|---|---|
| Esc-} | 1B, 7D | Disables the keyboard. |
| Esc-{ | 1B, 7B | Enables the keyboard. |
| Esc-v | 1B, 76 | Enables wrap-around at the end of the line. |
| Esc-w | 1B, 77 | Disables wrap-around at the end of the line. |
| Esc-z | 1B, 7A | Resets terminal to power-on configuration. |
| Esc-$ | 1B, 24 | Transmits the character at cursor location. |
| Esc-] | 1B, 5D | Transmits the 25th line. |
| Esc-# | 1B, 23 | Transmits the page. |
| Esc-( | 1B, 28 | Sets high intensity. |
| Esc-) | 1B, 29 | Sets low intensity. |
| Esc-+ | 1B, 2B | Clears the foreground. (High-intensity displayed characters.) |
| Esc-Z | 1B, 5A | Identifies display as emulating VT52 (the terminal responds with an Esc-K). |
| Esc-0 | 1B, 30 | Sets the underline mode. |

**continued**

| ESCAPE SEQUENCE | ASCII CODE GENERATED (HEXADECIMAL) | SEQUENCE DEFINITION |
|---|---|---|
| Esc-1 | 1B, 31 | Resets the underline mode. |
| Esc-2 | 1B, 32 | Enables cursor blink. |
| Esc-3 | 1B, 33 | Disables cursor blink. |
| Esc-8 | 1B, 38 | Sets the test (literally) mode for the next single character. |
| Esc-i[n] | 1B, 69 | Displays the system sign-on banner, as follows (n is the ASCII numeric character): |

| n | Display |
|---|---|
| 0 | Entire banner. |
| 1 | Company logo only. |
| 2 | Product name only. |
| 3 | Configuration information only. |

## 132-COLUMN UTILITY B.4

### Description

The 132-column utility (132C) provides a simulated 132-column display in the 800-dot by 400-line HIRES display mode of your computer. The characters are displayed in a 5-by-7 dot matrix in a 6-by-10 cell to give the 132-column display. In addition, a standard display of 80 columns by 25 lines is simulated with an 8-by-11 dot matrix in a 10-by-16 cell. The 132 column mode is enabled when the normal BIOS display interface recognizes the appropriate escape sequence.

### Installation

Enter this sequence to activate the 132 Column feature for use with an application program:

**A>132C<cr>**
**A>132C-ON<cr>**

These commands can be combined in a Batch (.BAT) file, if you like.

The 132C feature is turned off with the command:

**A>132C-OFF** <cr>

The 132C program installs itself into the BIOS. The program requires about 50K of memory: the 800-by-400-line HIRES screen requires 40K; the character sets and code use the rest. The 132C program copies itself into the lowest 64K block of memory and removes its 50K required bytes from the system until you press the reset button or turn off your computer.

You can access 132C after installing an application program if you send an ESC | (1B 7C) to the display interface. This is done from MS-BASIC with:

**PRINT CHR$(27)+CHR$(124)**

From MS-BASIC, 132C is turned off with an ESC z (1B 7A):

**PRINT CHR$(27)+"z"**

The display control sequences described above are also available in 132C mode.

When you've finished and saved your work, press the reset button to release the memory allocated to 132C and return to normal character mode.

14
3
57

# APPENDIX C

## ASCII Character Codes

| Dec | Hex | CHR | Dec | Hex | CHR | Dec | Hex | CHR |
|---|---|---|---|---|---|---|---|---|
| 000 | 00H | NUL | 043 | 2BH | + | 086 | 56H | V |
| 001 | 01H | SOH | 044 | 2CH | , | 087 | 57H | W |
| 002 | 02H | STX | 045 | 2DH | – | 088 | 58H | X |
| 003 | 03H | ETX | 046 | 2EH | . | 089 | 59H | Y |
| 004 | 04H | EOT | 047 | 2FH | / | 090 | 5AH | Z |
| 005 | 05H | ENQ | 048 | 30H | 0 | 091 | 5BH | [ |
| 006 | 06H | ACK | 049 | 31H | 1 | 092 | 5CH | \ |
| 007 | 07H | BEL | 050 | 32H | 2 | 093 | 5DH | ] |
| 008 | 08H | BS | 051 | 33H | 3 | 094 | 5EH | ^ |
| 009 | 09H | HT | 052 | 34H | 4 | 095 | 5FH | _ |
| 010 | 0AH | LF | 053 | 35H | 5 | 096 | 60H | ` |
| 011 | 0BH | VT | 054 | 36H | 6 | 097 | 61H | a |
| 012 | 0CH | FF | 055 | 37H | 7 | 098 | 62H | b |
| 013 | 0DH | CR | 056 | 38H | 8 | 099 | 63H | c |
| 014 | 0EH | SO | 057 | 39H | 9 | 100 | 64H | d |
| 015 | 0FH | SI | 058 | 3AH | : | 101 | 65H | e |
| 016 | 10H | DLE | 059 | 3BH | ; | 102 | 66H | f |
| 017 | 11H | DC1 | 060 | 3CH | < | 103 | 67H | g |
| 018 | 12H | DC2 | 061 | 3DH | = | 104 | 68H | h |
| 019 | 13H | DC3 | 062 | 3EH | > | 105 | 69H | i |
| 020 | 14H | DC4 | 063 | 3FH | ? | 106 | 6AH | j |
| 021 | 15H | NAK | 064 | 40H | @ | 107 | 6BH | k |
| 022 | 16H | SYN | 065 | 41H | A | 108 | 6CH | l |
| 023 | 17H | ETB | 066 | 42H | B | 109 | 6DH | m |
| 024 | 18H | CAN | 067 | 43H | C | 110 | 6EH | n |
| 025 | 19H | EM | 068 | 44H | D | 111 | 6FH | o |
| 026 | 1AH | SUB | 069 | 45H | E | 112 | 70H | p |
| 027 | 1BH | ESCAPE | 070 | 46H | F | 113 | 71H | q |
| 028 | 1CH | FS | 071 | 47H | G | 114 | 72H | r |
| 029 | 1DH | GS | 072 | 48H | H | 115 | 73H | s |
| 030 | 1EH | RS | 073 | 49H | I | 116 | 74H | t |
| 031 | 1FH | US | 074 | 4AH | J | 117 | 75H | u |
| 032 | 20H | SPACE | 075 | 4BH | K | 118 | 76H | v |
| 033 | 21H | ! | 076 | 4CH | L | 119 | 77H | w |
| 034 | 22H | " | 077 | 4DH | M | 120 | 78H | x |
| 035 | 23H | # | 078 | 4EH | N | 121 | 79H | y |
| 036 | 24H | $ | 079 | 4FH | O | 122 | 7AH | z |
| 037 | 25H | % | 080 | 50H | P | 123 | 7BH | { |
| 038 | 26H | & | 081 | 51H | Q | 124 | 7CH | \| |
| 039 | 27H | ' | 082 | 52H | R | 125 | 7DH | } |
| 040 | 28H | ( | 083 | 53H | S | 126 | 7EH | ~ |
| 041 | 29H | ) | 084 | 54H | T | 127 | 7FH | DEL |
| 042 | 2AH | * | 085 | 55H | U |  |  |  |

Dec=decimal, Hex=hexadecimal (H), CHR=character.
LF=Line Feed, FF=Form Feed, CR=Carriage Return, DEL=Rubout

# INDEX